# 第8章 常量

# Const的用途

- 值替代
- const与指针
- 在函数的参数和返回值中使用const
- 在类中使用const

# 值替代

- 在用C语言进行程序设计时，预处理器可以不受限制地建立宏并用它来替代值。
- 由于预处理只进行文本替代，没有类型检查的概念和相关功能，预处理器的值替代会产生一些微小的问题，这在C++中可以通过使用const避免。

# 值替代

- 下面的宏定义：
  - #define BUFSIZE 100

  BUFSIZE是一个名字，只是在预处理期间存在。
- C++用const改写上述表达：
  - const int bufsize = 100;
  - char buf[bufsize];
- 编译器的优化处理

# 头文件里的const

- 如果想在多个文件中使用const,需要将const定义放进头文件中：
  - const int bufsize = 100;
- 通常情况下C++编译器不为const常量创建存储空间，只是将该值保存在符号表中，进行常量替换。而在下述情况下,将强制编译器进行存储分配。
  - extern const int bufsize;

# const的安全性

- const的作用不仅限于在常数表达式里代替#defines。对于一个const值，如果在程序中试图改变它，编译器会给出出错信息。
- C08:Safecons.cpp

# 集合

- const可以用于集合，但编译器不会复杂到把一个集合保存到它的符号表中，所以必须分配内存。
  - const int i[] = {1,2,3,4};
- 在这种情况下，const意味着“不能改变的一块存储空间”
- 不能在编译期间使用集合内部的值，因为编译器在编译期间不知道他们的值。
- C08:Constag.cpp

```
1  //: C08:Constag.cpp
2  // From Thinking in C++, 2nd Edition
3  // Available at http://www.BruceEckel.com
4  // (c) Bruce Eckel 2000
5  // Copyright notice in Copyright.txt
6  // Constants and aggregates
7  const int i[] = { 1, 2, 3, 4 };
8  //! float f[i[3]]; // Illegal
9  struct S { int i, j; };
10 const S s[] = { { 1, 2 }, { 3, 4 } };
11 //! double d[s[1].j]; // Illegal
12 int main() {} ///:~
```

# 与C语言的区别

- 在C++中，是常量，一个const不一定创建内存空间
- 在C中，“an ordinary variable that cannot be changed”，一个const总是需要创建一块内存空间。
  - const int bufsize = 100;
  - char buf[bufsize ];
- C标准中，const定义的常量是全局的，C++中视声明位置而定。

# 指针

- 指向const的指针
  - 指向的内容不可改变
- const指针
  - 指针指向的地址不可改变
- 指针指向的地址和内容都不可改变

# 指向const的指针

- const int* u;
  - u是一个指针，它指向一个const int。
  - **const** specifier binds to the thing it is “closest to”

- int const * u

# const指针

- int d = 1;
- int* const w = &d;
  - **w** is a pointer, which is **const**, that points to an **int**
  - *w = 2是合法的。
- 定义一个指向const对象的const指针：
  - Int d = 1;
  - const int* const x = &d;
  - int const* const x2 = &d;
  - 在这种情况下，指针和对象都不能改变。

# 赋值和类型检查

- 编译器禁止将一个const对象的地址赋给非const指针。
- 除非程序员进行强制类型转换。
- C08:PointerAssignmet.cpp

```cpp
//: C08:PointerAssignment.cpp
// From Thinking in C++, 2nd Edition
// Available at http://www.BruceEckel.com
// (c) Bruce Eckel 2000
// Copyright notice in Copyright.txt
int d = 1;
const int e = 2;
int* u = &d; // OK -- d not const
//! int* v = &e; // Illegal -- e const
int* w = (int*)&e; // Legal but bad practice
int main() {} ///:~
```

# 函数参数和返回值

- const可以用来修饰函数的参数。
- 有两种情况
  - 传递和返回const值
  - 传递和返回const地址

# 传递const值

- 可指定按值传递的参数是const的

```
void f1(const int i){
        i++; //illegal
}
```

```
void f2(int ic){
        const int& i = ic;
        i++; //illegal
}
```

# 返回const值

- 如果一个函数返回的类对象为const时，那么这个函数的返回值不能是一个左值，不能被赋值，不能修改。
- C08:ConstReturnValues.cpp

- 临时量
  - 在求表达式值期间，编译器必须创建临时对象，尽管他们与正常对象一样有生命周期，但程序员从来看不到他们：编译器决定他们的去留。
  - 他们自动地被声明为常量。

# 传递和返回const地址

- 如果传递或返回一个地址(一个指针或一个引用)，客户程序使用该地址修改对象是可能的。如果这个指针或引用声明为const，就会阻止修改对象的操作。
- 无论什么时候传递一个地址给一个函数，就尽可能用const修饰它。
- C08:ConstPointer.cpp

# 类里的const

- 类里的常量
- 编译期间类里的常量
- Const对象
- Const成员函数

# 类里的常量

- 在类中定义一个常量的含义是：在这个对象的生命周期内，它是一个常量，然而，对这个常量而言，每个不同的对象可以含有一个不同的值。
- 在一个类中声明一个普通的常量时，不能给它初值，要在构造函数的一个特别的地方进行初始化：初始化列表。
- 初始化列表保证初始化代码一定在其他代码执行之前被执行。
- C08:ConstInitialization.cpp

```
 7 #include <iostream>
 8 using namespace std;
 9
10 class Fred {
11   const int size;
12 public:
13   Fred(int sz);
14   void print();
15 };
16
17 Fred::Fred(int sz) : size(sz) {}
18 void Fred::print() { cout << size << endl; }
19
20 int main() {
21   Fred a(1), b(2), c(3);
22   a.print(), b.print(), c.print();
23 } ///:~
```

```
 9 class B {
10   int i;
11 public:
12   B(int ii);
13   void print();
14 };
15
16 B::B(int ii) : i(ii) {}
17 void B::print() { cout << i << endl; }
18
19 int main() {
20   B a(1), b(2);
21   float pi(3.14159);
22   a.print(); b.print();
23   cout << pi << endl;
24 } ///:~
```

```
#include <iostream>
using namespace std;

class Integer {
  int i;
public:
  Integer(int ii = 0);
  void print();
};

Integer::Integer(int ii) : i(ii) {}
void Integer::print() { cout << i << ' '; }

int main() {
  Integer i[100];
  for(int j = 0; j < 100; j++)
    i[j].print();
} ///:~
```

# const对象和成员函数

- 使用const可以声明对象在其生命周期内是不会被改变的：
  - const blob b(2);
- const对象只能调用const函数，也就是不会改变对象状态的函数
  - int f() const;
- C08:ConstMember.cpp
  - 在const对象上调用非const函数的示例

# const对象和成员函数

| 数据成员 | 非const的普通成员函数 | const成员函数 |
| --- | --- | --- |
| 非const的普通数据成员 | 可以引用，也可以改变值 | 可以引用，不可以改变值 |
| const数据成员 | 可以引用，不可以改变值 | 可以引用，不可以改变值 |
| const对象 | 不允许 | 可以引用，不可以改变值 |

# 按位const和按逻辑const

- 如果想要建立一个const成员函数，但仍然希望在对象里改变某些数据，这关系到按位const(bitwise)和按逻辑const(logical)
- 按位const
  - 对象中的每个字节都是固定不变的.
- 按逻辑const
  - 虽然整个对象从概念上将是不变的，但可以成员为单位指定可改变的部分

# 修改const对象的两种方式

- 使用mutable关键字
  - 使用mutabel，可以指定一个特定的数据成员可以在一个const对象中改变。
  - Mutable.cpp
- 强制转换常量性
  - 通过强制转换去除其常量特性
  - C08:Castaway.cpp

# Const作用

- 1可以定义const常量
- 2便于进行类型检查
  - const常量有数据类型，而宏常量没有数据类型。编译器可以对前者进行类型安全检查，而对后者只进行字符替换，没有类型安全检查
- 3可以保护被修饰的东西
  - 防止意外的修改，增强程序的健壮性。
- 4可以很方便地进行参数的调整和修改
  - 同宏定义一样，可以做到不变则已，一变都变

# Const作用

- 5可以节省空间，避免不必要的内存分配
- 6提高了效率
  - 编译器通常不为普通const常量分配存储空间，而是将它们保存在符号表中，这使得它成为一个编译期间的常量，没有了存储与读内存的操作，使得它的效率也很高

# volatile

- volatile的含义是：“在编译器认识的范围外，这个数据可以被改变”
- volatile告诉编译器不要做出任何有关该数据的假定，比如通过将该数据读入寄存器优化。
- C08:volatile.cpp

# 回顾

- 如何定义常量
- 如何定义常量指针
- 如何定义常量对象
- 类中成员定义为常量
  - 常量方法
  - 常量属性